Programming Assignment 5: Numpy

This is an individual lab. YOU MUST HAVE YOUR OWN SUBMISSION. You need to submit the `numpy_intro.py` as a .**txt file** and screenshots of your output **as a pdf or jpeg (whatever your TA prefers)**. I will deduct points for those who do not follow instructions.

**Background:**
Numpy is a powerful tool that many programmers use for matrix and vector computation. In this lab, you will go through this tutorial to get an introduction to numpy.

**Part 0: Numpy Installation**
Run `numpy_intro.py`. If you are getting an error because of line 1 (`import numpy as np`), then this means that you do not have numpy installed yet. Please follow the instructions to install numpy: https://numpy.org/install/
** I recommend using pip or conda

**Part 1: Looking at the file**
Assuming that numpy is installed and ready to run, let's take a look at the file. You should see a 2d list called kobe.

```
kobe = [[18, 7.6], [19, 15.4], [20, 19.9], [21, 22.5], [22, 28.5], [23, 25.2],
        [24, 30], [25, 24], [26, 27.6] , [27, 35.4], [28, 31.6], [29, 28.3],
        [30, 26.8], [31, 27], [32, 25.3], [33, 27.9], [34, 27.4], [35,13.8],
        [36, 22.3], [37, 17.6]]
```

Please note that this is a LIST! We have not made it into a numpy array yet.

**Part 2: Changing list into a numpy array**
To change a list into a numpy array, we want to use the function:

`numpy.array(list_name)`

In the `numpy_intro.py` file, convert the `kobe` list into a numpy array called `kobe_np`. Please note that we imported the numpy library as np. You will have to use `np.array(list_name)` to convert the list into a numpy array instead.

To test that your code worked, try the following code:

```
print(kobe_np)
print(type(kobe_np))
```

Your output should look like this:

```
[[18.   7.6]
 [19.  15.4]
 [20.  19.9]
 [21.  22.5]
 [22.  28.5]
 [23.  25.2]
 [24.  30. ]
 [25.  24. ]
 [26.  27.6]
 [27.  35.4]
 [28.  31.6]
 [29.  28.3]
 [30.  26.8]
 [31.  27. ]
 [32.  25.3]
 [33.  27.9]
 [34.  27.4]
 [35.  13.8]
 [36.  22.3]
 [37.  17.6]]
<class 'numpy.ndarray'>
```

Hopefully, you can see that we have a matrix (a n-dimensional numpy array). Try the following code

```
print(kobe_np.shape)
```

You should see that it is a 20 x 2 matrix (20 rows and 2 columns). The shape attribute is a tuple where the first element is the number of rows and the second element is the number of columns. Instead of printing it out, I want you to save the number of rows and columns into variables called `num_rows` and `num_cols` respectively.
Hint: use brackets [] like you would in a list to get specific elements in a tuple

**Part 3: Introducing some numpy functions:**
Please make sure to comment out the previous print statements to prevent confusion with the output.

We are going to cover the following numpy functions in this section:

- Transpose
- Make vector of ones
- Stacking two numpy arrays together
- Concatenate two numpy arrays
- Matrix Multiplication
- Inverse of Matrix

**Part 3a: Transpose:**
Recall what transpose means: the rows become the columns and the columns become the rows. To find the transposed matrix, use the following:

```
matrix_name.T
```

Your task for 3a is the transpose the `kobe_np` matrix and store it as `kobe_transpose`.
To test it, print `kobe_transpose`. You should see something like this:

```
[[18.  19.  20.  21.  22.  23.  24.  25.  26.  27.  28.  29.  30.  31. 32.  33.  34.  35.  36.  37. ]
 [ 7.6 15.4 19.9 22.5 28.5 25.2 30.  24.  27.6 35.4 31.6 28.3 26.8 27. 25.3 27.9 27.4 13.8 22.3 17.6]]
```

The columns have become rows and the rows have become columns.

**Part 3b: Make vector of ones**
Numpy has a useful function where you can make vector (1d numpy array) of specific dimension filled with ones. You do not need to do it manually using a loop. This function is:

```
np.ones(dimension)
```

Where dimension is the number of ones you want in your vector.
Your task is to make a vector called ones using the function above. The dimension that you want to use here is `num_rows` from part 1. In this case you should have a 1d-numpy array with 20 ones in it.

**Part 3c: Accessing specific elements in a numpy array**
To get specific parts of a numpy array, you use brackets [] as you would in a list. However, there is a minor difference. Try the following code:

```
print(kobe_np[1])
print(kobe_transpose[1])
print(kobe_np[1, 0])
print(kobe_transpose[1, 0])
```

```
[[18.   7.6]
 [19.  15.4]
 [20.  19.9]
 [21.  22.5]
 [22.  28.5]
 [23.  25.2]
 [24.  30. ]
 [25.  24. ]
 [26.  27.6]
 [27.  35.4]
 [28.  31.6]
 [29.  28.3]
 [30.  26.8]
 [31.  27. ]
 [32.  25.3]
 [33.  27.9]
 [34.  27.4]
 [35.  13.8]
 [36.  22.3]
 [37.  17.6]]
```

**Kobe_np**

```
[[18.  19.  20.  21.  22.  23.  24.  25.  26.  27.  28.  29.  30.  31. 32.  33.  34.  35.  36.  37. ]
 [ 7.6 15.4 19.9 22.5 28.5 25.2 30.  24.  27.6 35.4 31.6 28.3 26.8 27. 25.3 27.9 27.4 13.8 22.3 17.6]]
```

**Kobe_transpose**

Hopefully, you can see what each did.
`matrix_name[#]` gets a specific row
`matrix_name[r, c]` gets a specific value at row r, column c

Your task for this section is to get the following vector:
`[18. 19. 20. 21. 22. 23. 24. 25. 26. 27. 28. 29. 30. 31. 32. 33. 34. 35. 36. 37.]`
And store it into a variable called `A`

Also, get this vector:
`[ 7.6 15.4 19.9 22.5 28.5 25.2 30.  24.  27.6 35.4 31.6 28.3 26.8 27. 25.3 27.9 27.4 13.8 22.3 17.6]`
And store it into a variable called `y`

**Part 3d: Concatenate two numpy arrays**

There are multiple ways to combine two numpy arrays. However, we are going to focus on

`numpy.column_stack((col1, col2))`

vstack concatenates two 1-d numpy arrays to make a 2-d numpy array. Go ahead stack `A` and `ones` in that order. Store it in a variable called `x`. When you print it, it should look like…

```
[[18.  1.]
 [19.  1.]
 [20.  1.]
 [21.  1.]
 [22.  1.]
 [23.  1.]
 [24.  1.]
 [25.  1.]
 [26.  1.]
 [27.  1.]
 [28.  1.]
 [29.  1.]
 [30.  1.]
 [31.  1.]
 [32.  1.]
 [33.  1.]
 [34.  1.]
 [35.  1.]
 [36.  1.]
 [37.  1.]]
```

**Part 3e: Matrix Multiplication**

Two multiply two matrices, you can use the following function (remember the precondition for matrix multiplication. If the precondition is not met, you will get an error)

```
numpy.matmul(mat1, mat2)
```

Try to multiply `x`'s transpose and `x` (IN THAT ORDER) and store it as `x_prod`. You should get a `2x2` matrix.

**Part 3f: Finding inverse of matrix**

To find the inverse of the matrix, you can use the following function

```
numpy.linalg.inv(matrix_name)
```

Find the inverse of `x_prod` and store it as `x_prod_inv`

**Part 4: On your own**

So far, you have calculated

$$x\_prod\_inv = (x^T x)^{-1}$$

Now, I want you to calculate the following

$$theta = (x^T x)^{-1}(x^T y)$$

Your theta should be a 2-vector

**Part 5: Your submission**

Your output should include the following:

- `x_prod_inv`
- `theta[0]`
- `theta[1]`

Your submission should be a screenshot of what is printed out on your console. Everything should be labeled! Sample screenshot below